ELEN 685 Lab 3

Checking for Palindromes

Learning Objectives

This lab will give you practice with a number of LC-2 assembly programming constructs. In particular you will cover the following topics:

- Load/store usage
- Addressing modes
- Looping constructs
- Register comparisons
- Condition codes
- Assembly language programming

Prerequisites

You should be familiar with the following concepts:

- ASCII representation of letters
- Writing pseudo-code

Description

You are to write a program, in LC-2 assembly language, that detects palindromes. A palindrome is a word or phrase that is spelled the same forwards as it is backwards, e.g. madam, racecar. We will fix the size of the word to check at exactly 10 characters. Your program will check the 10-character string starting at memory location x3200 to see if it is a palindrome. If the string is a palindrome, you are to store a 1 in memory location x3400. If it is not a palindrome, you are to store a 0 in memory location x3400. So if the string to be checked is x4345665434, your program must determine that this is a palindrome and set x3400 to 1. If the string to be checked is x4345664034, your program must determine that this is NOT a palindrome and set x3400 to 0. Each character in the string can be any printable ASCII character. One character is stored per memory word, even though an ASCII character only requires seven bits of storage; the high-order 9 bits of each memory word must contain zeroes. Additionally, your program must contain a loop to perform the letter comparisons instead of repeating the code some number of times.

Prelab Assignment

Data Movement Instructions

Data movement instructions move information between memory and registers. The LC-2 supports three modes by which data can be moved between memory and registers. These are direct, indirect, and base+offset. Each mode is available for both loads and stores.

Recall from lecture that a load operation moves data from a memory location to a register and a store moves data from a register to a memory location. The LC-2 provides seven instructions that move data. There are three load operations (LD, LDR, and LDI), and three store operations (ST, STR, STI). The prefixes LD and ST indicate load and store, respectively, while no suffix indicates the direct addressing mode, I indicates indirect, and R indicates base+offset. The LEA instruction is also considered a data movement instruction, even though it does not involve moving data between memory and a register. Instead, it calculates an address relative to the current page (as in direct mode) and stores the address in a register. (This is "immediate" mode, since the instruction's operand - the address - is specified by the instruction itself.)

Direct

Direct mode (LD and ST) is useful for referencing memory locations on the same page as the load instruction. (An LC-2 page is a 512-word region of memory.) Since the memory address is specified in the instruction, the location of the data being referenced will be the same each time the instruction is executed. In other words, the address is not data-dependent. Direct mode is useful when a constant value is needed to initialize a register, but the value is too large to be specified with immediate mode.

For example:

```
                LD R3, LoopCt ; lots of loop iterations
Loop            ADD R3, R3, #-1
                BRz  DONE
                    ...
                BRnzp  Loop
DONE            TRAP x25
LoopCt          .FILL 10000    ; too big to use imm mode
```

Another use is for temporary storage. For example, we may need to save the value of a register while we use it for something else. We can allocate some memory nearby to save and restore the register's value.

```
        ST  R0, SaveR0 ; save R0
        IN       ; get char (in R0)
          .
          .
          .
        LD  R0, SaveR0  ; done with char, restore R0
        BRnzp  NEXT
SaveR0  .BLKW  1
NEXT
```

The disadvantage of direct mode is that we may only reference locations that are in the same page. To access the rest of memory, we must use either base+offset or indirect mode.

Base+Offset

Base+offset addressing (LDR and STR) calculates the address by adding a small unsigned index (specified in the address) to the contents of a register. Since a register can contain any value, this mode can be used to address any location in memory. In the following example, the data to be loaded (at address x4000) is on a different page than the instruction (at x3001).

```
        .ORIG  x3000
        LD  R5, FarData ; use direct mode to load address
        LDR  R2, R5, 0   ; use base+offset to load data
          ...
        FarData  .FILL  x4000
```

(We could also use indirect mode for this example - see below. Base+offset mode is a better choice if we will be accessing the data many times, since it avoids the extra memory read.)

The use of a register also makes it easy to process a sequence of memory locations. In the following example, we are calculating a table of powers of two. We use the LEA instruction to set R2 to the first location in the table. On each loop iteration, we increment R2 to store into the next location.

```
        LEA  R2, TwosTable
        AND  R0, R0, 0  ; initialize R0 to 1
        ADD  R0, R0, 1
        AND  R1, R1, 0   ; initialize R1 (loop counter) to 8
        ADD  R1, R1, 8
Loop    STR  R0, R2, 0   ; write next table entry
        ADD  R0, R0, R0 ; compute next power of two
        ADD  R2, R2, #1  ; point to next table entry
        ADD  R1, R1, #-1
        BRp  Loop
        BRnzp  Next
TwosTable .BLKW  8  ; storage for 8-element table
Next            ...
```

Indirect

With indirect mode (LDI and STI), the address specified by the instruction is not the location to be accessed. Instead, it contains the address of the data to be accessed.

Therefore, two memory accesses are required: first, a memory read to get the address; second, a memory read or write to transfer data from/to memory.

As with base+offset, indirect mode can also be used to access any location memory, not just within the same page, since the contents of the first memory location can be any 16-bit value. The first address, however, must be within the same page, since it is specified using a page offset (just like direct mode). Here is the same example used above, accessing data on a different page, this time using indirect mode:

```
        .ORIG  x3000
        LDI  R5, FarData ; use indirect mode to load data
          ...
FarData  .FILL  x4000
```

Exercises (30 pts)

1. (20 pts) Complete the exercises on the last page of this laboratory. Include your name and turn in the page to your lab instructor at the beginning of the lab period.

(10 pts) Think of two or more methods for checking for palindromes. There are always multiple ways to solve a problem. Each of these methods should be handwritten accounts in the form of pseudocode ("first do this, then do that," etc.) or a flow chart (boxes, circles, lines indicating decisions, etc.) of what your program does. The accounts should show the step-by-step operations needed for the programs, including any loops and branches. Your methods should be individual work, handwritten on a piece of paper. Include your name. Your TA will check off your work (on the prelab sheet) when you attend lab.

Laboratory Assignment
Palindrome Checker Program (70 pts)

Enter your palindrome checker program, in assembly language (".asm"), using the LC-2 editor. Remember that the first line of your program should tell the simulator where to load the program. The assembly language instruction that loads the program at beginning at memory location x3000 is ".ORIG x3000". In addition, your assembly language program must end with the directive ".END" on the last line of the file. (This does not take the place of the "TRAP x25" that you've used in previous programs. It merely tells the assembler to stop processing the file.) Make sure to include comments in your code.

Save your file with an appropriate file name, for example "palindrome.asm." Recall that you must convert your program to true binary, creating a palindrome.obj file that the simulator understands. Once you've converted your program, load it in to the simulator and test it. If changes need to be made in the program, make the changes in the .asm file, then re-assemble and reload the program.

When using the simulator, you will need to enter the string to be checked, starting at memory location x3200. To do this, type x3200 in the "Jump to:" box and press Enter. Then find the line starting with x3200 in the display (it should be the first line). Then double click anywhere on the line with the x3200. That will bring up a "Set Value" dialog. For the value, type in the ASCII value, in hex, that you wish, e.g. x63 for the letter `c'. Then press ok. You can click on the next 9 memory locations and set the values for the letters you want. Be sure to check that your program works for strings that are palindromes and for those that are not.

Before submitting your program, place a header embedded in comments at the beginning of your file. The header should include information needed by someone trying to use your program, or at least to figure out what it does. You must include the program name, your name, section number, date, a description of your program, and any assumptions that you have made in writing your program. Below is a sample header. Recall that comments can be added to an assembly language file by preceding them with a semicolon.

```
;
; PROG:  divider.asm
;
; NAME:  John Doe
; SEC:   223
; DATE:  Oct. 4, 2000
;
; DESC:  divider.asm is an assembly language program that divides
;        a positive 15-bit integer by another positive 15-bit
;        integer to produce a 15-bit quotient and remainder.
;
; ASMPT: The two integers are 15-bit positive 2's complement
;        integers and are stored in registers R0 and R1 before
;        program execution. The result produced is that of R0
;        divided by R1.
;

.ORIG x3000             ; tells the LC-2 where to load the program
.
.
.
.END                    ; indicates that this is the end of program
```

Name: _____

Prelab Exercises

1. Data movement instructions (14 pts)
Fill in the missing parts of the instructions and/or interpretations below. For target address, specify (a) the address of the data for direct mode (as in "x5807"), (b) the base register and offset for base+offset mode (as in "R7 + 9"), or (c) the address of the load/store address for indirect mode (as in "x5807").

Address	Instruction	Operation	Src/Dst	Target Address
x3100	0110101010101001	_____	_____	_____
x5F29	1010_____	_____	R2	x5E11
x220C	____001_____	STR	_____	R6+34
x_____	0011110_____	_____	R6	x6A77
x7F36	____010011111111	LDR	R2	_____
x3340	0111100101101101	_____	_____	_____

2. Choice of addressing mode (6 pts) For each of the following scenarios, specify the most appropriate data movement instruction: LD, LDR, LDI, LEA, ST, STR, or STI.

(a) Write a value into a memory location 1000 words away from the current instruction location. This location will only be accessed a few times over the lifetime of the program.

(b) A load instruction is part of a loop, accessing a different location (depending on program data) for each loop iteration.

(c) A register must be saved to a temporary memory location (and will be restored shortly thereafter).

3. Programming alternatives (10 pts) - on a separate piece of paper _____